Université Paris 13 | INFO2-MI1
Institut Galilée | Année 2011-2012

# Programmation Linéaire
# TP n°4 : Cplex Callable Library

## Exercice 1

Écrire un programme C/C++ permettant de résoudre le programme linéaire suivant :

$$\begin{cases} Max & -5x_1 - 7x_2 + 8x_3 \\ s.c. & 3x_1 + 2x_2 \leq 9 \\ & 10x_1 + 2x_3 \geq -11 \\ & x_1 \leq 0 \\ & 1 \leq x_2 \leq 4 \\ & 0 \leq x_3 \leq 3 \end{cases}$$

Vous utiliserez pour cela la Cplex Callable Library et en particulier la fonction `CPXloadlp`. Vous prendrez soin de bien lire la documentation qui vous est fournie.

**Utilisation de la bibliothèque d'optimisation Cplex**

Cplex inclut une bibliothèque, développée en trois langages (C : libcplex.a, C++ : libilocplex.a, Java : cplex.jar), et qui contient un certain nombre de fonctions qu'il est possible d'appeler dans un programme (C, C++, Java). Dans le cadre de ce TP, nous travaillerons exclusivement en C.

# 1. Utilisation de la bibliothèque Ilog Cplex

## 1.1. Description

La bibliothèque Ilog Cplex contient des routines organisées en plusieurs catégories :

1. les routines d'*optimisation* qui permettent de résoudre un problème ;
2. les routines de *modélisation* qui permettent de (re)définir un problème ;
3. les routines de *paramétrisation* qui servent à connaître ou à modifier les valeurs des paramètres des fonctions de Cplex ;
4. les routines d'*interrogation* qui permettent d'accéder aux informations inhérentes à un problème et à sa solution.

## 1.2. Documentation technique

### 1.2.1 Organisation et conventions de nommage

Les structures et fonctions Cplex ont pour préfixe "CPX". Les fichiers d'entête (essentiellement, **cplex.h** pour la bibliothèque C) contiennent les constantes (ex. : CPX_INFBOUND pour désigner l'infini) et les prototypes de toutes les routines et paramètres de Cplex. Ces fichiers sont situés dans le répertoire : **/LOCAL/cplex-10.1/cplex101/include/ilcplex**

### 1.2.2 Compilation

Pour faire appel aux objets Cplex, il faut bien évidemment :

1. dans vos sources, penser à inclure **cplex.h** : **#include<cplex.h>**
2. dans la commande de compilation, indiquer les chemins (fichier d'entêtes et bibliothèque) et le lien à la bibliothèque (NB : il faut aussi utiliser les librairies standard **m** et **pthread**) :

```
-I/LOCAL/cplex-10.1/cplex101/include
-L/LOCAL/cplex-10.1/cplex101/lib/x86_RHEL3.0_3.2/static_pic -lcplex -lm -lpthread
```

### 1.2.3 Initialisation de l'environnement Cplex

Pour invoquer des objets ou fonctions Cplex, il faut en premier lieu initialiser l'environnement, Cplex ayant besoin de quelques structures de données internes pour opérer. La routine **CPXopenCplex()** permet cette initialisation et retourne un pointeur vers l'environnement créé (dont le type est **CPXENVptr**). Toutes les structures et fonctions Cplex nécessitent la référence à un environnement.

Notons que plusieurs environnements (autant que de licences disponibles) peuvent être ouverts simultanément. Le mode d'emploi de **CPXopenCplex()** est le suivant :

| | |
|---|---|
| CPXENVptr CPXPUBLIC CPXopenCplex (int *statut); | |
| **CPXENVptr** : | type Cplex qui désigne un pointeur vers l'environnement |
| **statut** : | contient le code de l'erreur éventuelle (0 en cas de succès). |

Ainsi, pour ouvrir l'environnement Cplex, il faut déclarer une variable de type **CPXENVptr**, déclarer une variable de type **int**, puis affecter l'environnement par appel à **CPXopenCplex()** :

```
CPXENVptr env = NULL; int statut = 0;
env = CPXopenCplex (&statut);
```

### 1.2.4 Gestion de l'erreur

La plupart des fonctions retournent un entier : 0 en cas de succès, une certaine constante symbolique en cas d'erreur. Pour retrouver la description associée à un code numérique donné, il faut déclarer une chaîne de caractères de taille suffisante, puis faire appel à la fonction **CPXgeterrorstring** ; le message correspondant à l'entier **statut** est écrit dans une chaîne (**str_err_message** dans l'exemple ci-suit) :

```
str_err_message[1024];
CPXgeterrorstring (env, statut, str_err_message);
```

### 1.2.5 Libération de l'espace mémoire

Il faut libérer l'espace réservé par les structures Cplex. Malheureusement, il n'existe pas de fonction permettant de libérer toute réservation mémoire faite pour des structures liées à un certain environnement ; il faut donc *d'abord* libérer les différentes structures (notamment, les programmes linéaires par la fonction **CPXfreeprob()**), puis l'environnement Cplex par la fonction **CPXcloseCPLEX()** :

```
// libérer l'espace occupé par un problème
int CPXPUBLIC CPXfreeprob (CPXCENVptr env, CPXLPptr *lp_p);
// libérer l'espace occupé par un environnement
int CPXPUBLIC CPXcloseCPLEX (CPXENVptr *env_p);
```

# 2. Manipulation d'un programme linéaire

## 2.1. Définition

Une fois un environnement ouvert, il faut initialiser une structure de donnée dans laquelle un problème à résoudre sera stocké. Cette initialisation se fait par appel à la routine **CPXcreateprob()**, qui retourne un pointeur vers un PL :

| | |
|---|---|
| CPXLPptr CPXPUBLIC CPXcreateprob(CPXCENVptr env, int* status_p, const char*& probname_str); | |
| **CPXLPptr :** | type Cplex qui désigne un pointeur vers une structure PL ; |
| **probname_str :** | désigne le nom donné au problème. |

Plusieurs problèmes peuvent être ouverts simultanément dans un même environnement. Après son initialisation, il est possible de définir un problème de plusieurs manières : par structures informatiques, par appels successifs à des fonctions de modification de PL, ou encore, par lecture de fichier.

## 2.2. Initialisation

### 2.2.1 Par structures

On peut rassembler toutes les données dans différents tableaux et appeler la routine **CPXcopylp()** qui va copier les données dans la structure de donnée PL :

```
int CPXPUBLIC CPXcopylp
(
        CPXCENVptr env, CPXLPptr lp,
        int numcols, int numrows, int objsen,
        const double *obj, const double *rhs, const char *sense,
        const int *matbeg, const int *matcnt, const int *matind, const double *matval,
        const double *lb, const double *ub, const double *rngval
);
```

**numcols, numrows, objsen** :
nombre de colonnes, nombre de lignes, sens de l'objectif ($\in$ {CPX_MIN, CPX_MAX}).
**obj, rhs, sense** :
vecteur objectif, vecteur membre droit, vecteur de sens des contraintes ($\in$ {'L', 'E', 'G'}).
**matbeg, matcnt, matind, matval** :
vecteurs représentant la matrice des contraintes.
**lb, ub** :
vecteurs des bornes inférieures et supérieures pour les variables.
**rngval** : pas dit dans ce TP.

Seuls les coefficients *non nuls* de la matrice des contraintes sont représentés, dans le tableau **matval** ; **matdeb** associe à chaque colonne de la matrice des contraintes l'indice du premier élément à lire dans **matval** ; **matind** associe à chaque coefficient de **matval** son indice (i.e., sa ligne) dans la colonne à laquelle il appartient ; enfin, **matcnt** associe à chaque colonne le nombre de coefficients non nuls qu'elle contient. Ainsi, les coefficients non nuls de la colonne $j$ sont :

**matval[ matdeb[j] ], matval[ matdeb[j]+1 ], . . ., matval[ matdeb[j]+matcnt[j]-1 ]**

et se situent respectivement dans la ligne :

**matind[ matdeb[j] ], matind[ matdeb[j]+1 ], . . ., matind[ matdeb[j]+matcnt[j]-1 ]**

### 2.2.2 Par fonctions

On peut appeler successivement les routines **CPXnewcols()**, **CPXnewrows()**, **CPXaddcols()**, **CPXaddrows()** (*cf.*, partie ) : soit l'on définit le problème par ses variables (**CPXnewcols()**) puis on remplit la matrice en décrivant totalement les contraintes (**CPXaddrows()**), soit l'on définit le problème par ses contraintes (**CPXnewrows()**) puis on remplit la matrice en décrivant totalement les variables (**CPXaddcols()**).

### 2.2.3 Par lecture de fichier

Si les données sont déjà rangées dans un fichier aux formats **lp** ou **mps**, la routine **CPXreadcopyprob()** permet de lire le fichier et d'en copier les données dans la structure PL :

int CPXreadcopyprob(CPXENVptr env, CPXLPptr lp, char *nom_fichier, char *type_fichier);

## 2.3. Interrogation

Plusieurs accesseurs sont naturellement définis qui permettent de récupérer les données d'un problème, les plus importants étant sans doute ceux permettant d'en connaître la taille :

```
// pour récupérer le nombre de variables de lp
int CPXPUBLIC CPXgetnumcols (CPXCENVptr env, CPXCLPptr lp);
// pour récupérer le nombre de contraintes de lp
int CPXPUBLIC CPXgetnumrows (CPXCENVptr env, CPXCLPptr lp);
```

Pour les autres accesseurs (objectif, membre droit, lignes, colonnes, bornes, ...), se reporter à l'aide Cplex (ne serait-ce, que **cplex.h**).

## 2.4. Modification d'un problème

### 2.4.1 Ajout de ligne (contrainte), de colonne (variable)

#### Création de lignes et de colonnes incomplètes

Par incomplètude, on entend absence des coefficients correspondants dans la matrice des contraintes. Une contrainte est alors caractérisée par le membre droit et le sens, tandis qu'une variable est caractérisée par son coefficient dans la fonction objectif, son type (continu, binaire, ...) et ses bornes (inférieure et supérieure). Les nouvelles lignes ou colonnes créées sont ajoutées à la suite (du point de vue de leur indice) des lignes et colonnes déjà existantes.

```
int CPXPUBLIC CPXnewrows        // ajout de contraintes vides à un problème
(
    CPXCENVptr env, CPXLPptr lp,
    int nb, const double *rhs, const char *sense, const double *rngval, char **rowname
);
```

| | |
|---|---|
| **nb** : | nombre de contraintes |
| **rhs** : | vecteur (de taille **nb**) des membres droits |
| **const char *sense** : | vecteur (de taille **nb**) des sens des contraintes |

```
int CPXPUBLIC CPXnewcols        // ajout de variables à un problème
(
    CPXCENVptr env, CPXLPptr lp,
    int nb, const double *obj, const double *lb, const double *ub, const char *xctype, char **colname
);
```

| | |
|---|---|
| **nb** : | nombre de variables |
| **obj** : | vecteur (de taille **nb**) des coefficients objectif |
| **lb, ub** : | vecteurs (de taille **nb**) des bornes inf. et sup. |
| **const char *xctype** | vecteur des types des variables (réel == 'C', binaire == 'B', entier == 'I', ...) |

Une ligne (*resp.*, une colonne) ainsi créée peut ensuite être complétée par appel à la fonction **CPXaddcols()** (*resp.*, **CPXaddrows ()**), ou encore, aux fonctions de modification des coefficients, telles que **CPXchgcoef()** ou **CPXchgcoeflist()** (documentées plus loin au paragraphe ).

#### Création de lignes et de colonnes complètes

Il faut donner les mêmes informations que précédemment, avec en plus la description des coefficients de la matrice. Les fonctions sont **CPXaddrows()** et **CPXaddcols()**. De nouveau, les lignes ou

colonnes insérées sont ajoutées à la suite (du point de vue de leur indice) des lignes et colonnes déjà existantes. **Attention** : autant l'on peut ajouter implicitement une variable (donc, une colonne) en ajoutant une ligne (cas d'une contrainte faisant intervenir une variable qui n'était jusque lors impliquée dans aucune contrainte existante), autant l'on ne peut ajouter de contrainte (considérer un indice de ligne supplémentaire) au moment de l'ajout d'une variable.

```
int CPXaddrows        // ajout de lignes complètes à un problème
(
        CPXCENVptr env, CPXLPptr lp,
        int ccnt, int rcnt, int nzcnt, const double *rhs, const char *sense,
        const int *rmatbeg, const int *rmatind, const double *rmatval,
        char **colname, char **rowname
);
```

**ccnt** : nombre de contraintes ajoutées
**rcnt** : nombre de variables (éventuellement) ajoutées
**nzcnt** : nombre de coefficients non nuls à insérer dans la matrice

```
int CPXaddcols        // ajout de colonnes complètes à un problème
(
        CPXCENVptr env, CPXLPptr lp,
        int ccnt, int nzcnt, const double *obj
        const int *cmatbeg, const int *cmatind, const double *cmatval,
        const double *lb, const double *ub, char **colname
);
```

### 2.4.2 Suppression de ligne (contrainte), de colonne (variable)

Les fonctions **CPXdelrows** et **CPXdelsetrows** (*resp.*, **CPXdelcols** et **CPXdelsetcols**) permettent de supprimer un ensemble de lignes (*resp.*, de colonnes) :

| |
|---|
| int CPXdelrows(CPXCENVptr env, CPXLPptr lp, int begin, int end)<br>int CPXdelcols(CPXCENVptr env, CPXLPptr lp, int begin, int end)<br>**begin, end** : indice des première et dernière lignes/colonnes à supprimer |
| int CPXdelsetrows(CPXCENVptr env, CPXLPptr lp, int *delstat)<br>int CPXdelsetcols(CPXCENVptr env, CPXLPptr lp, int *delstat)<br>**delstat** : vecteur de taille **CPXgetnumrows(env,lp)**/**CPXgetnumcols(env,lp)** ;<br>les lignes / colonnes d'indice **ind** tel que **delstat[ind]** == **1** sont supprimées |

### 2.4.3 Modification de données du problème

Un ensemble de fonctions préfixées par **CPXchg** (principalement, **CPXchgcoef**, **CPXchgcoeflist**, **CPXchgbds**, **CPXchgobj**, **CPXchgrhs**, **CPXchgsense** **CPXchgcoef**, **CPXchgprobtype**) permettent de modifier :

1. les coefficients de la matrice, incluant ceux de la fonction objectif et du membre droit ;
2. les bornes sur les variables ;
3. le sens des contraintes ;
4. les propriétés du problème.

Fonctions **CPXchgcoef**, **CPXchgcoeflist** :

```
int CPXPUBLIC CPXchgcoef        // (re)définition d'un coefficient de la matrice
(
    CPXCENVptr env, CPXLPptr lp,int i, int j, double newvalue
);
```

| | |
|---|---|
| **i** : | indice de ligne (-1 pour l'objectif) |
| **j** : | indice de colonne (-1 pour le membre droit) |
| **newvalue** : | la valeur du coefficient |

```
int CPXPUBLIC CPXchgcoeflist        // (re)définition d'un ensemble de coefficients de la matrice
(
    CPXCENVptr env, CPXLPptr lp,
    int numcoefs, const int *rowlist, const int *collist, const double *vallist
);
```

| | |
|---|---|
| **numcoefs** : | nombre de coefficients à modifier |
| **rowlist** : | vecteur des **numcoefs** indices de ligne des coefficients à modifier |
| **collist** : | vecteur des **numcoefs** indices de colonne des coefficients à modifier |
| **vallist** : | vecteur des **numcoefs** nouveaux coefficients à affecter |

Pour les autres fonctions, se reporter à la documentation Cplex.

# 3. Résolution d'un problème

La résolution du problème se fait à l'aide d'une des routines d'optimisation suivantes (après que le problème ait été défini et initialisé !) :

| fonction | équiv. | Synopsis |
|---|---|---|
| **CPXlpopt()** | **optimize** | int CPXPUBLIC CPXlpopt (CPXCENVptr env, CPXLPptr lp); |
| **CPXprimopt()** | **primopt** | int CPXPUBLIC CPXprimopt (CPXCENVptr env, CPXLPptr lp); |
| **CPXdualopt()** | **tranopt** | int CPXPUBLIC CPXdualopt (CPXCENVptr env, CPXLPptr lp); |

## 3.1. Récupération de la solution

Après appel à une fonction d'optimisation, les informations relatives à la résolution d'un problème peuvent être récupérées par la routine **CPXsolution()** :

```
int CPXPUBLIC CPXsolution
(
    CPXCENVptr env, CPXCLPptr lp, int *lpstat_p,
    double *objval_p, double *x, double *pi, double *slack, double *dj
);
```

| | |
|---|---|
| **x, pi** : | variables primales, coûts réduits des variables primales |
| **pi, slack** : | variables duales, variables d'écart |

# 4. Sauvegarde des informations

Les fonctions **CPXwriteprob** et **CPXwritesol** permettent respectivement d'écrire un problème et la solution d'un problème dans un fichier :

| | | |
|---|---|---|
| int CPXPUBLIC CPXwriteprob | / | int CPXPUBLIC CPXwritesol |
| (CPXCENVptr env, CPXCLPptr lp, const char *filename_str, const char *filetype_str) | | |
| **filetype_str** : | type du fichier (nulle si l'extension est donnée dans le nom du fichier) | |

# Annexes

## A.1. Compilation

1. exo.c → exo.o
   gcc -c exo.c -o exo.o -I/LOCAL/cplex-10.1/cplex101/include

2. exo.o → exo
   gcc exo.o -o exo -L/LOCAL/cplex-10.1/cplex101/lib/x86_RHEL3.0_3.2/static_pic
   -lcplex -lm -lpthread

## A.2. Programme Cplex minimal

```
#include<cplex.h>

int main()
{
    // declarations
    CPXENVptr env = NULL; // environnement Cplex
    CPXLPptr lp = NULL; // PL Cplex
    int statut = 0; // statut technique des fonctions

    // ouverture de Cplex
    env = CPXopenCPLEX (&statut);

    // creation d'un PL
    lp = CPXcreateprob(env, &statut, "gros_probleme");

    // instanciation d'un PL
    statut = CPXreadcopyprob(env, lp, "mon_pb.lp", NULL);

    // resolution d'un PL
    statut = CPXprimopt(env, lp);

    // souvegarde de la solution dans un fichier
    statut = CPXwritesol(env, lp, "resolution.txt", NULL);

    // liberation de l'espace memoire
    statut = CPXfreeprob (env, &lp);
    statut = CPXcloseCPLEX (&env);

    // on s'en va !
    return 0;
}
```